DW_crc_p

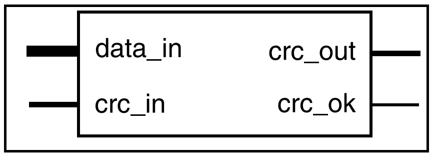
Universal Parallel (Combinational) CRC Generator/Checker

Version, STAR and Download Information: IP Directory

Features and Benefits

Revision History

- Parameterized arbitrary polynomial
- Parameterized data width
- Parameterized initial CRC value (all ones or all zeros)
- Parameterized inversion of generated CRC
- Parameterized bit and byte ordering



Description

DW_crc_p is a universal Cyclic Redundancy Check (CRC) Polynomial Generator/Checker that provides data integrity on data records in a single parallel calculation.

The CRC polynomial (size and coefficients) is specified at design time along with bit ordering, initial CRC, and CRC inversion characteristics. The data width is also specified at design time, and determines the number of data bits which DW_crc_p will be protecting.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
data_in	*data_width* bits	Input	Input data used for both generating and checking for valid CRC
crc_in	*poly_size* bits	Input	Input CRC value used to check a record (not used when generating CRC from `data_in`)
crc_ok	1 bit	Output	Indicates a correct residual CRC value, active high
crc_out	*poly_size* bits	Output	Provides the CRC check bits to be appended to the input data to form a valid record (`data_in` and `crc_in`)

Table 1-2 Parameter Description

Parameter	Values	Description
data_width	1 to 2560 Default: 16	Width of `data_in` (the amount of data that CRC will be calculated upon)
poly_size	2 to 64 Default: 16	Size of the CRC polynomial and thus the width of `crc_in` and `crc_out`

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
crc_cfg	0 to 7 Default: 7	CRC initialization and insertion configuration (see Table 1-5 on page 2)
bit_order	0 to 3 Default: 3	Bit and byte order configuration (see Table 1-6 on page 3)
poly_coef0	1 to 65535[a] Default: 4129[b]	Polynomial coefficients 0 through 15
poly_coef1	0 to 65535 Default: 0	Polynomial coefficients 16 through 31
poly_coef2	0 to 65535 Default: 0	Polynomial coefficients 32 through 47
poly_coef3	0 to 65535 Default: 0	Polynomial coefficients 48 through 63

a. *poly_coef0* must be an odd number (since all primitive polynomials include the coefficient 1, which is equivalent to X^0).
b. CCITT-CRC16 polynomial is X16 + X12 + X5 + 1, thus
poly_coef0 = 212 + 25 + 1 = 4129.

Table 1-3 Synthesis Implementations

Implementation Name	Implementation	License Feature Required
str	Synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW04.DW_CRC_P_CFG_SIM	Design unit name for VHDL simulation
dw/dw04/src/DW_crc_p_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_crc_p.v	Verilog simulation model source code

Table 1-5 CRC Configurations

crc_cfg	Virtual CRC Register Initial Value	Generated CRC bits
0	Zeros	Not Inverted
1	Ones	Not Inverted
2	Zeros	XORed with ... 010101

Table 1-5 CRC Configurations (Continued)

crc_cfg	Virtual CRC Register Initial Value	Generated CRC bits
3	Ones	XORed with ... 010101
4	Zeros	XORed with ... 101010
5	Ones	XORed with ... 101010
6	Zeros	Inverted
7	Ones	Inverted

Table 1-6 Bit Order Modes

bit_order	CRC Calculation Bit Order	Bytes Ordered[a]
0	MSB first, LSB last	MSB first, LSB last
1	LSB first, MSB last	LSB first, MSB last
2	MSB first, LSB last	LSB first, MSB last
3	LSB first, MSB last	MSB first, LSB last

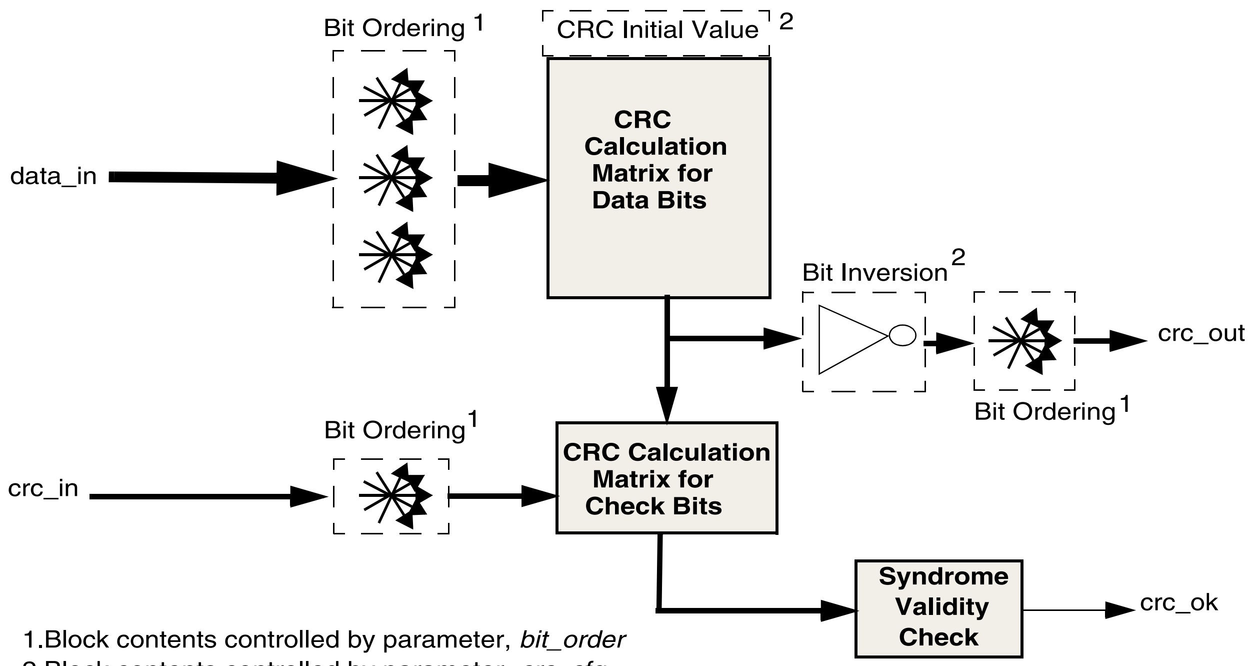
a. Byte ordering is only available when parameter *data_width* is a multiple of 8 and parameter *poly_size* is a multiple of 8. For *data_width* or *poly_size* values other than a multiple of 8, the *bit_order* value is restricted to 0 and 1.

Table 1-7 Parameters for Common CRC Implementations

Description	*poly_size*	*crc_cfg*	*poly_coef0*	*poly_coef1*	*poly_coef2*	*poly_coef3*
CRC-32	32	7	7607	1217	0 (not used)	0 (not used)
CRC-16	16	7	32773	0 (not used)	0 (not used)	0 (not used)
CCITT-CRC16	16	7	4129	0 (not used)	0 (not used)	0 (not used)
ATM Header CRC	8	2	7	0 (not used)	0 (not used)	0 (not used)
USB Token Packet CRC	5	7	5	0 (not used)	0 (not used)	0 (not used)

Figure 1-1 on page 4 shows the functional block diagram of DW_crc_p.

Figure 1-1 Functional Block Diagram of DW_crc_p



1.Block contents controlled by parameter, *bit_order*
2.Block contents controlled by parameter, *crc_cfg*

CRC Checkbit Generation

CRC checkbits are generated continuously at the output port, `crc_out`, based on the current data bits on the input port, `data_in`. If an instance of DW_crc_p is being used exclusively for checkbit generation, the input port, `crc_in`, is unnecessary and should be tied low (all zeros) while the output port, `crc_ok`, is also not needed and should be left unconnected.

CRC Checking

Data (on input port, `data_in`) and accompanying check bits (on input port, `crc_in`) are continuously monitored for validity as reflected by the value on the output port, `crc_ok`. If valid checkbits (for the accompanying data) are seen on the `crc_in` port, the `crc_ok` output port is driven active (high). Otherwise (such as for invalid checkbits of current data), `crc_ok` is driven inactive (low). If an instance of DW_crc_p is being used exclusively to check CRC, the output port, `crc_out`, is not needed and can be left unconnected.

Polynomial Specification

One of the most important aspects of a CRC error detection scheme is the size and quality of the generator polynomial.

Because CRC error detection has been used for many years, standard CRC generator polynomials of various sizes are widely used. In networking, CRC-32 is the most popular polynomial used for data payload protection.

Some standard protocols use CRC-16, while others use CCITT-CRC-16. In most cases, system architecture or inter-operability with standard protocols usually determine the polynomial used in an error detection system.

The generator polynomial of DW_crc_p is controlled by user-specified parameters (*poly_coef0*, *poly_coef1*, *poly_coef2*, and *poly_coef3*). Table 1-7 on page 3 lists the values of these parameters for several standard generator polynomials. If the generator polynomial required for a CRC system is not listed in Table 1-7 on page 3, the polynomial coefficient values can easily be calculated with the method described in the following section.

Calculating Polynomial Coefficient Values Specification

Each of the four polynomial coefficient values represent a sixteen-bit slice of polynomial coefficient positions. Thus, the following is true:

- Terms from 1 to x^{15} are contained in *poly_coef0*
- Terms from x^{16} to x^{31} are contained in *poly_coef1*
- Terms from x^{32} to x^{47} are contained in *poly_coef2*
- Terms from x^{48} to x^{63} are contained in *poly_coef3*

The term x^{poly_size} does not need to be contained in the polynomial coefficient values because it is implied by the value of `poly_size` itself (ex. for a 32-bit polynomial x^{32} is implicit and therefore does not need to be specified). Thus, for *poly_coef0*, add up all of the terms of the generator polynomial between 1 and x^{15}, with x = 2. For *poly_coef1*, sum all terms of the generator polynomial between x^{16} and x^{31}, with x =2, all divided by 2^{16}. For *poly_coef2*, sum all terms of the generator polynomial between x^{32} and x^{47}, with x = 2, all divided by 2^{32}. For *poly_coef3*, sum all terms of the generator polynomial between x^{48} and x^{63}, with x = 2, all divided by 2^{48}. This process is essentially a binary to decimal conversion with terms that appear in a polynomial being ones, and terms that do not appear in the polynomial being zeros.

The following example shows the coefficients for CRC-32:

CRC-32 = $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$
poly_coef0 = $x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$, evaluated at x = 2
poly_coef0 = $2^{12} + 2^{11} + 2^{10} + 2^{8} + 2^{7} + x^{5} + 2^{4} + 2^{2} + 2 + 1 =$
$4096 + 2048 + 1024 + 256 + 128 + 32 + 16 + 4 + 2 + 1$
poly_coef0 = 7607 (binary equivalent = 0001110110110111)
poly_coef1 = $(x^{26} + x^{23} + x^{22} + x^{16})/2^{16}$, evaluated at x = 2
poly_coef1 = $(2^{26} + 2^{23} + 2^{22} + 2^{16})/2^{16} = 2^{10} + 2^{7} + 2^{6} + 1 = 1024 + 128 + 64 + 1$
poly_coef1 = 1217 (binary equivalent = 0000010011000001)

Because *poly_size* = 32 for CRC-32, there are no terms beyond `x31`. Thus, *poly_coef2* `=` *poly_coef3* `= 0`.

Virtual CRC Register Configuration

Virtual CRC register configuration refers to the following:

- How the virtual register that calculates the CRC is initialized (to ones or zeros)
- Which (if any) bits are inverted when generating the CRC check bits

DW_crc_p has eight different virtual CRC register configurations as specified by the *crc_cfg* parameter. The most common initial value of virtual CRC registers is all ones, but there are protocols that use zeros as the initial virtual CRC register value. Odd values of *crc_cfg* e.g., 1, 3, 5, and 7) initialize the virtual CRC register to all ones, while even values (e.g., 0, 2, 4, and 6) initialize it to all zeros.

It is common to invert bits of the CRC check bits when inserting them into the data stream. DW_crc_p can be configured as follows:

- All CRC check bits are inverted when *crc_cfg* = 6 or 7
- Odd check bits (as in bits 1, 3, 5, etc.) are inverted when *crc_cfg* = 4 or 5
- Even check bits (as in bits 0, 2, 4, etc.) are inverted when *crc_cfg* = 2 or 3
- No check bits are inverted when *crc_cfg* = 0 or 1

Thus, the most common configuration (initialize to ones and invert CRC being inserted) is selected using 7 as the value of *crc_cfg*. One uncommon configuration can be found in the ATM networking standard's Header Error Check (HEC) field where the even bits are inverted and the register is initialized to all zeros (*crc_cfg* = 2.) Table 1-5 on page 2 shows all seven configurations.

Data Bit Order

CRC calculation is order dependent. A CRC system must calculate CRC values in the proper order to conform to a particular protocol and thus be interoperable.

DW_crc_p is configurable to calculate CRC values on data and check bits in one of four bit order configurations, as specified by the *bit_order* parameter. The *bit_order* parameter can be set to the simple most significant bit first (*bit_order* = 0) or least significant bit first (*bit_order* = 1) configuration regardless of the *data_width* and *poly_size* parameters. However, some protocols require a more intricate bit ordering scheme as they order bits within a byte in one direction (LSB to MSB), while bytes within a block are ordered in the opposite direction (MSB to LSB). Thus, *bit_order* values of 2 and 3 support such schemes, and are only valid when *data_width* and *poly_size* are an integer multiple of eight.

Figure 1-2 through Figure 1-5 show the bit order schemes for DW_crc_p.

Figure 1-2 Bit Order Scheme: *bit_order* = 0

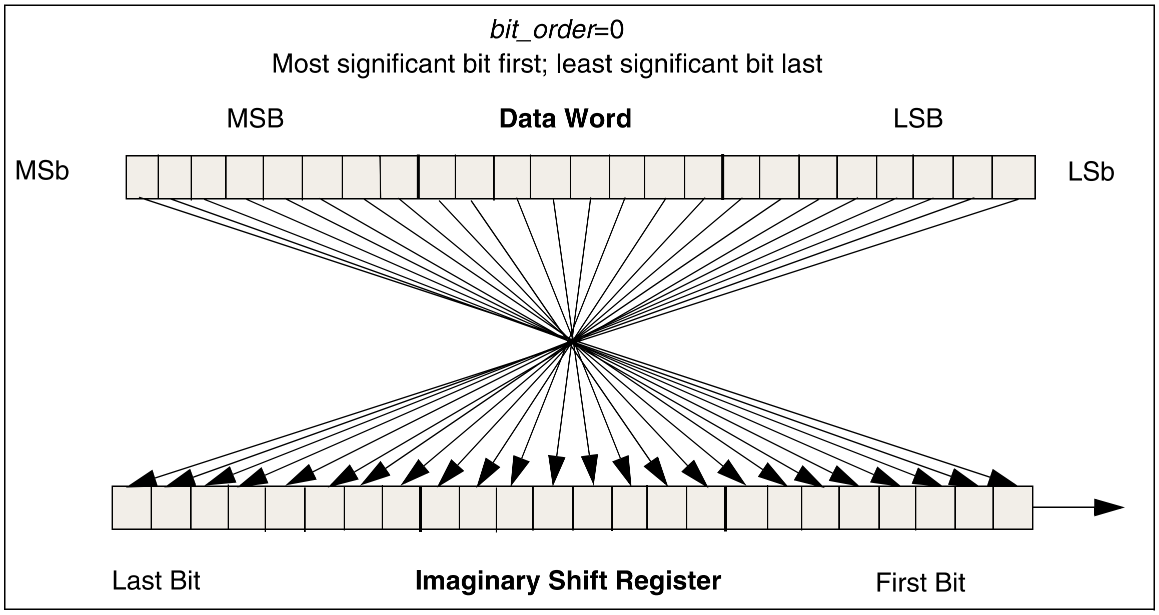

bit_order=0
Most significant bit first; least significant bit last
MSB
Data Word
LSB
MSb
LSb
Last Bit
Imaginary Shift Register
First Bit

Figure 1-3 Bit Order Scheme: *bit_order* = 1

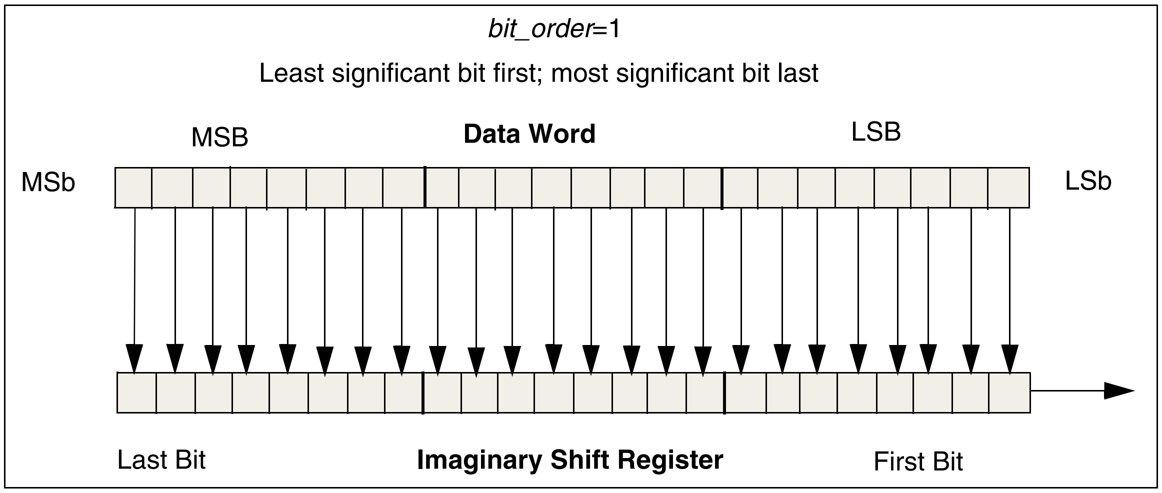

bit_order=1
Least significant bit first; most significant bit last
MSB
Data Word
LSB
MSb
LSb
Last Bit
Imaginary Shift Register
First Bit

Figure 1-4 Bit Order Scheme: *bit_order*=2

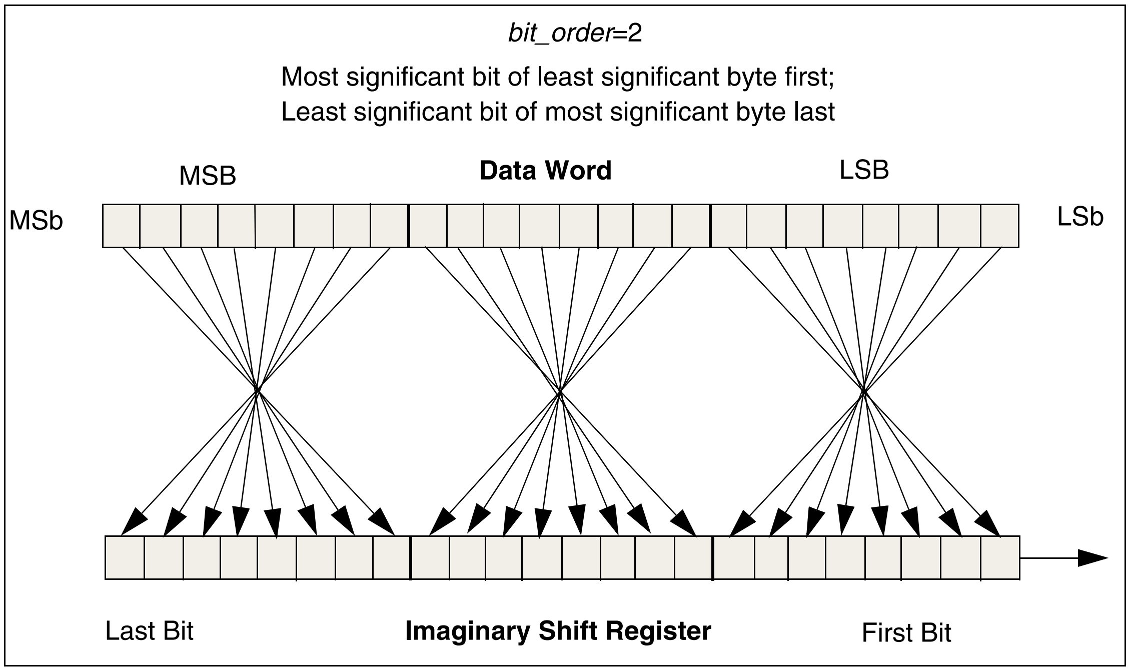
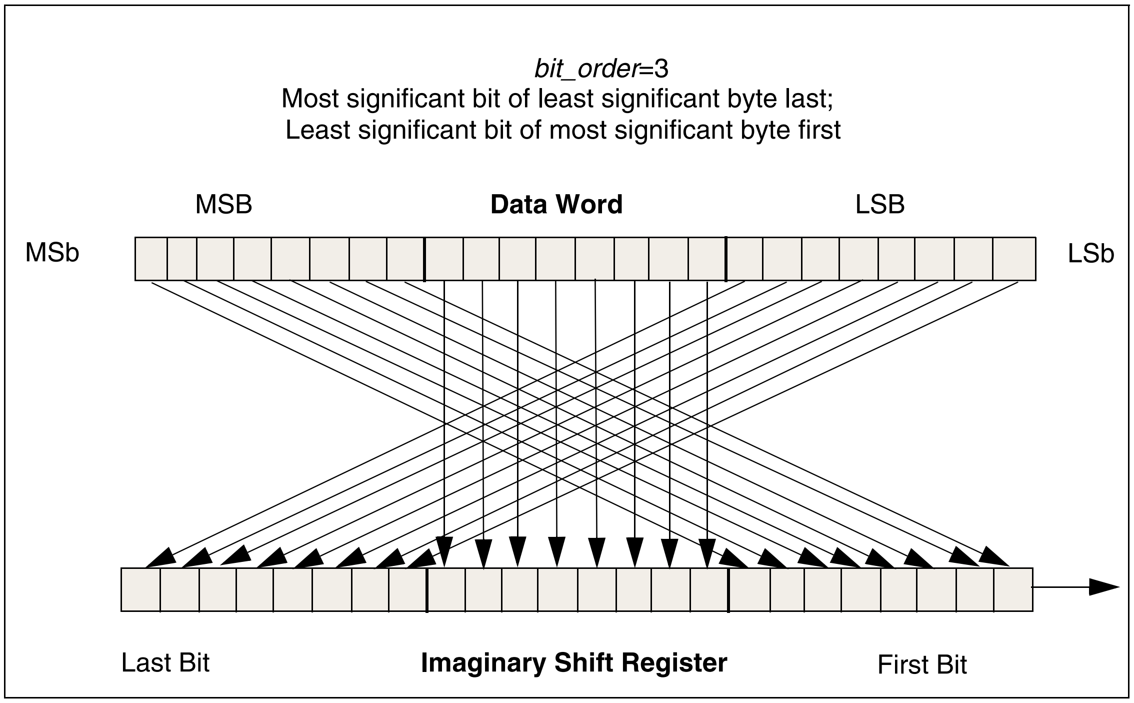

bit_order=2
Most significant bit of least significant byte first;
Least significant bit of most significant byte last
MSB
Data Word
LSB
MSb
LSb
Last Bit
Imaginary Shift Register
First Bit

Figure 1-5 Bit Order Scheme: *bit_order*=3


bit_order=3
Most significant bit of least significant byte last;
Least significant bit of most significant byte first
MSB
Data Word
LSB
MSb
LSb
Last Bit
Imaginary Shift Register
First Bit

Related Topics

- Application Specific – Data Integrity Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW_crc_p_inst is
  generic (inst_data_width : INTEGER := 19;  inst_poly_size   : INTEGER := 5;
           inst_crc_cfg    : INTEGER := 7;   inst_bit_order   : INTEGER := 0;
           inst_poly_coef0 : INTEGER := 5;   inst_poly_coef1 : INTEGER := 0;
           inst_poly_coef2 : INTEGER := 0;   inst_poly_coef3 : INTEGER := 0
           );
  port (inst_data_in : in std_logic_vector(inst_data_width-1 downto 0);
        inst_crc_in  : in std_logic_vector(inst_poly_size-1 downto 0);
        crc_ok_inst  : out std_logic;
        crc_out_inst : out std_logic_vector(inst_poly_size-1 downto 0)  );
end DW_crc_p_inst;

architecture inst of DW_crc_p_inst is
begin

  -- Instance of DW_crc_p
  U1 : DW_crc_p
    generic map (data_width => inst_data_width,  poly_size => inst_poly_size,
                 crc_cfg => inst_crc_cfg,        bit_order => inst_bit_order,
                 poly_coef0 => inst_poly_coef0,
                 poly_coef1 => inst_poly_coef1,
                 poly_coef2 => inst_poly_coef2,
                 poly_coef3 => inst_poly_coef3 )
    port map (data_in => inst_data_in,   crc_in => inst_crc_in,
              crc_ok => crc_ok_inst,    crc_out => crc_out_inst );
end inst;

configuration DW_crc_p_inst_cfg_inst of DW_crc_p_inst is
  for inst
  end for;
end DW_crc_p_inst_cfg_inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_crc_p_inst( inst_data_in, inst_crc_in, crc_ok_inst, crc_out_inst );

  parameter data_width = 16;
  parameter poly_size = 16;
  parameter crc_cfg = 7;
  parameter bit_order = 3;
  parameter poly_coef0 = 4129;
  parameter poly_coef1 = 0;
  parameter poly_coef2 = 0;
  parameter poly_coef3 = 0;

  input [data_width-1 : 0] inst_data_in;
  input [poly_size-1 : 0] inst_crc_in;
  output crc_ok_inst;
  output [poly_size-1 : 0] crc_out_inst;

  // Instance of DW_crc_p
  DW_crc_p #(data_width,   poly_size,   crc_cfg,   bit_order,   poly_coef0,
              poly_coef1,   poly_coef2,  poly_coef3)
    U1 (.data_in(inst_data_in),   .crc_in(inst_crc_in),         .crc_ok(crc_ok_inst),
    .crc_out(crc_out_inst) );
endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
June 9, 2021	DWBB_202106.0	■ Corrected explanation of the valid checkbits explanation in "CRC Checking" on page 4
October 2020	DWBB_202009.1	■ For STAR 3372033, updated the value of *data_width* Table 1-2 on page 1
January 2019	DWBB_201806.5	■ Updated example in "HDL Usage Through Component Instantiation - VHDL" on page 9 ■ Added this Revision History table and the document links on this page